Center for Advanced Computation

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

2/4-15

CAC Document No. 76

ALGORITHM FOR HIGH SECURITY CODES

by

George Purdy

June 6, 1973

CAC Document No. 76

ALGORITHM FOR HIGH SECURITY CODES

By

George Purdy

Center for Advanced Computation
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

June 6, 1973

This work was supported in part by the Advanced Research Projects Agency of the Department of Defense and was monitored by the U.S. Army Research Office-Durham under Contract No. DAHC04-72-C-0001.

ABSTRACT

In this report we describe a security coding system which enables the user to log onto a system by using his code number X_i which is immediately transformed into a pseudo code word $Y_i = f(X_i)$ by the machine. Even though Y_i and f are public knowledge it is not possible to log onto the system without knowing X, and the equation $Y = f(x)$ cannot be solved for X, even if Y is known.

THE SECURITY CODE

Some time in January, 1973 we were asked by Larry Roberts of Arpa to design a coding system to prevent unauthorized use of machines on the Arpa network. The problem somewhat loosely phrased, was to find a function f such that the user's password X would be transformed into Y, where $Y = f(X)$, so that X would not be stored in the machine and would therefore not be accessible to would be crackers of the system.

I suggested a method, which was immediately rejected by Larry Roberts on the grounds that the cracker could solve for X, given Y. It was only then that the actual problem became clear. Namely, it must be impossible to invert f, since f will be known to the cracker if the system has no read protection. I then suggested, and we later implemented a method using an f that could not practically be inverted.

Here is the description of the first method which I sent to Larry:

We think of the x_i as being code words for $1 \leq i \leq n$ and we shall think of the x_i as being integers between 1 and M. Typically we might expect n, the number of users to be about 1000 and M to be about 10^{40}. The Y_{ij} we think of as the time varying code for X_i at time t_i. A suggested time interval $t_i - t_{i-1}$ is one day.

Each $Y_{ij} = f_i(x_j)$, where $f_i(x)$ is the i^{th} of our permutations on $\{1, \ldots, M\}$. We make $f_i(x)$ have the property that $f_i(x)$ never equals x. A very convenient family of permutations is the family

$$f_i(x) = k_i x \pmod{P}$$

where P is a prime number very close to M (and we may need some computer time to find it) and k_i is the i^{th} random integer between 2 and P-1 (it

is very easy to produce good random k_i). We need an inverse permutation $g_i(x) = h_i\ x \pmod P$ where h_i is the mod P inverse of k_i and is found from k_i by Euclids algorithm at the time that k_i is generated. The program which calculates f and g will be very short; they will involve multi-precision arithmetic, but this is not too difficult to program. The effect of f and g will be completely unpredictable even to someone who has constant access to the Y_{ij} that are stored and P, because the k_i are produced by a random number generator.

How do we use the system? F will be used to check if X_i is the code of some account, to see how much money is left in account X_i, and to change account X_i for time used. The function G may be used at the end of the day (or other time interval) for reading off the account information into some more permanent record, if this is desired. The other purpose of G is to recode all of the Y_{ij} when the time interval changes from t_i to t_{i+1}. Actually, we would want the composition

$$f_{i+1}\ (g_i(x)) = k_{i+1}\ h_i\ x \pmod P$$

for that purpose, and of course $k_{i+1}\ h_i$ only has to be computed once mod P for each time interval.

We have some ideas about how the Y_{ij} should be stored, but there is no reason why they shouldn't be stored in the same way that the X_i are presently stored. It would be especially uncrackable, however, if the Y_{ij} were stored, along with their account information, by means of a HASH code and if the account information such as dollars available were coded by F's and G's also.

Because of the many details involved we would rather write the programs ourselves than send all of the many details to you.

This is the reply that I got from Larry Roberts:

Mon., Feb. 5, 1973

The system you proposed appears to store the remainder of the product of your password and K(I) mod P as the word to be recognized, Y. Thus:

$$Y(J) + N * P = X(J) * K(I)$$

I must assume for any system that a system cracker has full access to all info except X. Thus he need only proceed to use a linear diophantine solution technique on the above equation until he has X. All this can be done during a momentary penitration of system security. If I am correct a far better technique is needed, one which no one can crack even given all the info necessary to check the password E.G. The system operator should not be able to determine my password. I also can't see what added security is added by changing K each day. Any day I break in K is there along with all the Y's.

Please run any scheme you come up with past several good people to see if they can break it before considering programming it. Tell me if I misunderstood or about other ideas.

Larry Roberts

I then sent this note to Larry Roberts:

Date: Feb. 6, 1973 2:45 p.m.
From: Purdy
Re: Security System

You are right, the system we proposed does indeed store the remainder of the product of the password and K(I) mod P as the word to be recognized, Y(I,J).

$$Y(I,J) + N(I,J) * P = X(J) * K(I)$$

We are actually a little surprised, however, that a system cracker has full access to all info except X. We had imagined that the number K(I) could be kept in a reserved part of memory. If the user had access to K(I), then you are quite correct, K(I) could be computed using Euclid's algorithm, or he could apply Euclid's algorithm directly to the Diophantine equation.

From here on, let us assume therefore that the CRACKER has knowledge of everything except the password X.

The code will be Y = f(x) and the cracker will have knowledge of f. Then we arrange that f has no easily computable inverse. For example,

if f(x) = polynomial of degree 6 (mod P) then the inverse g of f, which of course is 6-valued, will probably not be computable by any reasonable algorithm. There is one algorithm for computing g which always exists--namely the cracker tries all possible code words X until one comes up for which f(X) = Y. There are P possible code words, so that if P is about 1.0E4.0 then it would take too long to do this. However for this to be true it is essential that the code words be truly random, and they will probably be hard to remember; there is no way to avoid this unpleasant aspect of the system if f is public information. To get around the non-uniqueness of the inverse of f(x), the code Y for the user code would be the vector (F(x), w) where the user code is (x, w). The word w only needs to be long enough to guarantee uniqueness and it is no help to the cracker, e.g. w might be the account number. (Since there are at least $P/6 \sim 10^{39}$ possible values for f(x), it is very unlikely that coincidences will occur in any case, so we might just do without w). Some time should be spent finding a suitable f.

It seems to us that this system satisfies your requirements, but there is always the possibility that we have not understood the problem entirely.

I then got this reply from Larry Roberts:

Date: Feb. 8, 1973 1914
From: Larry Roberts
Re: Security System

You now understand the problem. A penetrater easily makes himself a wheel and then accesses everything, but for a short while. I hope you can find a function soon since we are pushing for full security in the NET in the next few months. In some of the systems without special hardware all system files are readable, but not writeable by the users. Here such a security system is mandatory.

Don't ignore complex boolean functions (nonlinear) since these should be more profitable.

Hope to hear from you soon.
Larry Roberts

Then Larry Roberts came to visit and we thrashed out a few details and I eventually sent him the following:

In what follows, we describe a security coding system which enables the user to log onto a system by using his code number X_i which is immediately transformed into a pseudo code word $Y_i = f(X_i)$ by the machine. Even though Y_i and f are public knowledge it is not possible to log onto the system without knowing X, and the equation $Y = f(x)$ cannot be solved for X, even if Y is known.

§1 the function f(X)

The code is of the form $Y = f(X)$, and the cracker knows f. We have arranged that the equation $f(X) = Y$ is very unlikely to be solved in fewer than 10^6 seconds of processor time. The function f is a polynomial modulo a prime P.

$$f(X) = X^n + a_4 X^m + a_3 X^3 + a_2 X^2 + a_i X + a_o \pmod{P},$$

where $P = 2^{64} - 59$, $m = 2^{24} + 3$, $n = 2^{24} + 17$,

and the a_i are 19-digit numbers. The cracker has essentially two approaches for solving $Y = f(X)$ given Y. He can use trial and error, or Berlekamp's and similar algorithms. It was necessary to make n and P fairly large in order to defeat both approaches.

§2 Berlekamp's Algorithm as a Threat

Berlekamp's method for completely factoring polynomials modulo P can be applied to the polynomial $f(X) - Y = g(X)$ and it requires [1] at least $n^3 (\log P)^2$ operations. There are no algorithms known which are faster than this, so it seems safe to say that $n^2 (\log P)^2$ operations are required to find just a single root.

Now $n \cong 10^7$ and $P \cong 10^{19}$, so more than 10^{16} operations are required.

Let us say that the speed s of the crackers machine is 10^{10} operations per second (faster than ILLIAC IV). Then it would still take more than $T = 10^6$ seconds $\simeq$ two weeks.

§3 The Trial and Error Threat

We assume that the cracker has a list of all assigned Y_i and he keeps trying values of X until $f(X) = Y_i$ for some i. Let c be the number of X_i assigned to users. A theorem of Lagrange guarantees that no more than n of the X's will map into one Y. Thus, if the cracker chooses an X at random between 1 and P, his probability of success is at most $\frac{cn}{P}$.

The probability P_k of failure on the kth trial is at least

$$1 - \frac{cn}{P - k + 1} \approx 1 - \frac{cn}{P} = a.$$

The expected number of trials K_e before success is

$$K_e = \sum_{k=1}^{\infty} K\ (P_1\ P_2\ \ldots\ P_k) = \sum_{k=1}^{\infty} k\ a^k = \frac{a^2}{(1 - a)^2} \simeq \frac{P^2}{c^2 n^2}\ .$$

The expected cracking time is $T_e = \frac{K_e Q}{s} = \frac{P^2 Q}{c^2 n^2 s}$ where Q is the number of operations needed to compute f(X).

Even if $Q = 1$, and $s = 10^{10}$, $c = 1000$, we have

$$T_e \simeq \frac{10^{38}}{10^6\ 10^{14}\ x\ 10^{10}} = 10^8$$

seconds, or about 3 years.

§4 Implementation

The implementation of the algorithm for f uses multi-precision arithmetic in the form of some Fortran subroutines. It is operational on a PDP-10 and requires about half a second to computer f(X).

§5 Remarks about the use of the code

When user-codes are assigned, a random number generator should be used to choose an X_i and then one should verify that $f(X_i) \neq Y_j$ for any previous Y_j. This is extremely unlikely, but it could happen.

[1] D. E. Knuth, The Art of Computer Programming, Vol. 2, pp. 381-397, Addison-Wesley, 1969.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Center for Advanced Computation University of Illinois at Urbana-Champaign Urbana, Illinois 61801	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP

3. REPORT TITLE

Algorithm for High Security Codes

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Research Report

5. AUTHOR(S) *(First name, middle initial, last name)*

George Purdy

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 6, 1973	8	0

8a. CONTRACT OR GRANT NO. DAHC04-72-C-0001 b. PROJECT NO. ARPA Order No. 1899 c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) CAC Document No. 76 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. DISTRIBUTION STATEMENT

Copies may be requested from the address given in (1) above.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	U. S. Army Research Office - Durham Duke Station, Durham, North Carolina

13. ABSTRACT

In this report we describe a security coding system which enables the user to log onto a system by using his code number X_i which is immediately transformed into a pseudo code word $Y_i = f(X_i)$ by the machine. Even though Y_i and f are public knowledge it is not possible to log onto the system without knowing X, and the equation $Y = f(x)$ cannot be solved for X, even if Y is known.

DD FORM 1 NOV 65 1473

UNCLASSIFIED

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Supervisory Systems Utility Programs						